平成 20 年度前期 情報画像工学実験 II
実験 7. プロセッサの基礎

計算機システム工学教育研究分野 難波 一輝
1 号棟 409 室, 内線 3255
e-mail: namba@faculty.chiba-u.jp

実験内容

本テーマによる実験は 2 週間行われる。実験内容は以下の通りである。

第 1 週 マイクロプロセッサの動作

- スロースキャンコンピュータを用いたマイクロプロセッサの動作の理解と考察
- 4 号棟 2 階 203 号室にて、当日指定する班に分かれて行う。

第 2 週 アセンブラ言語

- アセンブラ言語の理解とアセンブラ言語による簡単なプログラミング
- 1 号棟 5 階 503 号室にて、各自それぞれで行う。

なお、本実験は計算機システム序論の教科書「入門計算機システム」[1] 、特に第 3 章、について、既に理解しているものとして構成されている。よく復習しておくこと。また、本実験は本テキストを充分予習しているものとして時間配分してある。よく予習しておくこと。

レポート提出

第 1 週目のレポート課題は本テキスト中に含まれる。第 2 週目のレポート課題については第 2 週目に使用するツールの説明と共に以下の Web サイト上で説明する。

`http://www.icsd2.tj.chiba-u.jp/~namba/lecture/`

なお、実験レポートの提出は 1 回のみである。2 週分の内容をまとめて、指定された時間までに 1 号棟 5 階 501 室前レポートボックスに提出すること。

大切なことは、課題をこなしながら実験テーマを理解することである。レポートには、課題に対する結論のみを示すのではなく、そこから何が分かったかなど、考察を行うこと。

[1] 伊藤, 倉田, "—," 朝倉出版, 2000

実験 7. プロセッサの基礎: 1 週目

— マイクロプロセッサの動作 —

1 実験の目的

CPU はコンピュータ内で基本的な演算処理を行う、いわばコンピュータの心臓部に当たる装置であり、マイクロプロセッサは CPU を 1 個の半導体チップに集積した部品である。マイクロプロセッサとしては Intel 社の Pentium シリーズや AMD 社の Athlon シリーズなどの x86 系プロセッサが有名であり、現在市場をほぼ独占している。x86 系の他に Apple 社の Macintosh に搭載されている PowerPC シリーズ、ワークステーションによく用いられる Sun Microsystems 社の SPARC シリーズなどがある。

マイクロプロセッサのアーキテクチャ(構造)はその種類によって異なる。しかし、その根本原理はどれも等しく、互いのアーキテクチャの類似性も高い。よって、1 種のマイクロプロセッサについて学べば、他のプロセッサについて学ぶことはさほど困難ではない。

本実験では（株）デンジニア社の製造した教育用マイクロプロセッサ、スロースキャンコンピュータ DSC-11 型を用い、プロセッサの動作について学ぶ。

2 スロースキャンコンピュータ

スロースキャンコンピュータは、ハードウェアの動きが理解できるようにコンピュータ内の各所の信号レベルを LED で表示しながら計算を行うことができる教育用コンピュータである。このコンピュータは、クロック周波数が 1Hz と 4Hz の切り替えとなっており、さらにスイッチを用いたステップ動作も可能なため、信号の流れを確認しながらプログラム実行の様子を見てゆくことができる。本章では本実験で用いるスロースキャンコンピュータ DSC-11 型について説明する。

2.1 ハードウェア

スロースキャンコンピュータのパネルを図 1 に示す。右側の上にヒューズがあり、その下には電源スイッチがある。電源スイッチの上側を押すとスロースキャンコンピュータの電源が入り上にあるパワーランプが点灯する。実験を終了する場合には必ず電源スイッチで電源を切ってから電源コードを抜くこと。

トグルスイッチが多数ついているが、重要なのは右側の 7 つである。これらはそれぞれ次のような意味を持っている。

- CLEAR
  表示用ランプをすべて消灯してコンピュータを初期状態に戻す。ただし、メモリの内容は変更されない。
- STOP
  RUN
  STOP 側に倒すとプログラムは停止する。RUN 側に倒すとプログラムは再開する。
- STEP
  AUTO
  プログラム実行方式をステップ実行 (STEP) とするか、自動で実行 (AUTO) するかを選択する。
- ADDRESS
  DATA
  メモリへの書き込みを行う際、指定しているデータがアドレスなのか (ADDRESS)、データ自身なのか (DATA) を表す。
- READ
  WRITE
  このスイッチを READ 側に倒すことで、現在プログラムカウンタに指定されている番地のメモリの内容が出力レジスタに表示される。このスイッチを WRITE 側に倒した場合、ADDRESS / DATA スイッチが ADDRESS となっていれば、入力レジスタで指定した値の下位 5 ビットがプログラムカウンタに代入される。DATA となっている場合には、入力レジスタで指定した値が、現在プログラムカウ

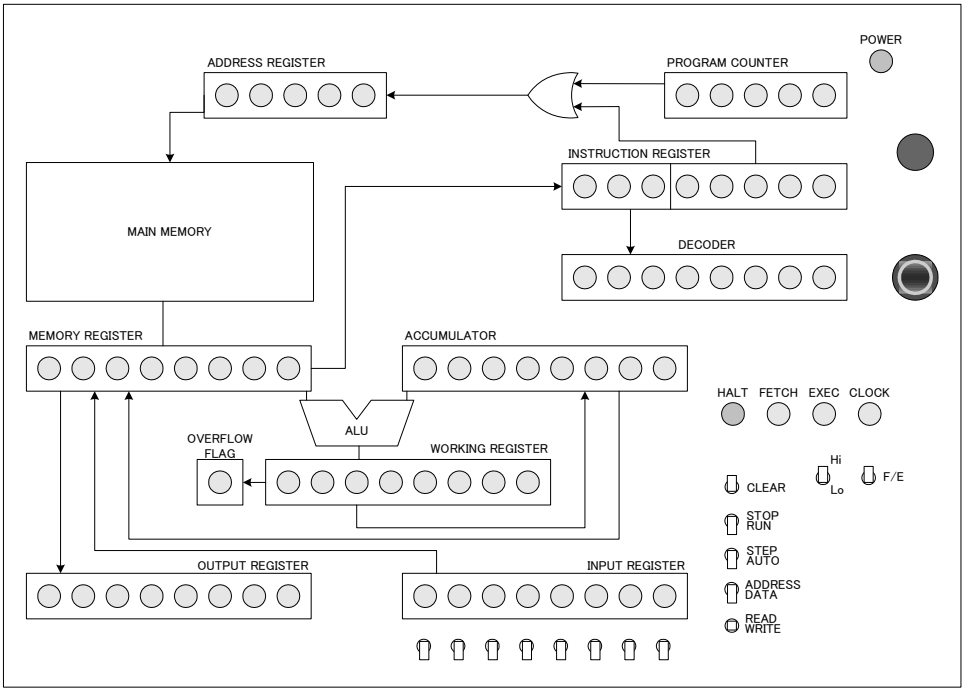

図 1: スロースキャンコンピュータ

ンタで指定されている番地のメモリに書き込まれる。

- Hi
  Lo
  プログラムを実行する際のクロックの周波数を 4Hz (Hi) と 1Hz (Lo) とで切り替える。
- F/E
  STEP
  AUTO
  スイッチが STEP となっている場合、実行フェーズ（EXEC LED が点灯）から次の取り出しフェーズ（FETCH LED が点灯）への移行、またはその逆を行う。

入力レジスタの下にあるスイッチは入力レジスタの値を指定するためのものであり、上に倒すと対応するビットが 1 に、下に倒すと 0 になる。

2.2 ソフトウェア

スロースキャンコンピュータは、さまざまな演算を施すための ALU を持つ。ALU の入力はメモリレジスタとアキュムレータに接続されている。したがって、各種演算はアキュムレータとメモリレジスタ、つまり、あるメモリの内容との演算が主となる。メモリは 32 ワードが搭載されており、0〜31 までの番地を利用できる。スロースキャンコンピュータには、図 1に挙げる J（分岐命令）、A（加算命令）、B（減算命令）、L（ロード命令）、T（ストア命令）、R（入力命令）、W（出力命令）、S（シフト命令）の 8 つの命令が存在する。

分岐命令の特殊形態である停止命令 (J/0) はアキュムレータの値には関係なく実行を停止する。このとき、スロースキャンコンピュータは HALT ランプを点灯して実行を停止する。READ 命令を実行した場合、プログラムの実行をいったん中断し、入力レジスタからの値の読み込みを行う。入力レジスタの下のスイッチ（上向きは 1、下向きは 0）を操作して入力値を設定したら、その右にある READ / WRITE スイッチを下に倒すことで値を入力する。WRITE 命令を実行した場合、出力レジスタの LED が値のビットパターンどおりに点灯する（1 が点灯、0 は消灯）。出力レジスタの様子は、新たな値を出力するまでそのままとなる。

演算対象の値の入力などでは、10 進数ではなく 2 進数で値を指定しなければならない。入力

表 1: スロースキャンコンピュータの命令

形式	コード	命令の意味
J/n	000	n が 0 の時はプログラムの実行を停止する アキュムレータの値が 0 以上のとき n 番地に分岐する[1] アキュムレータの値が 0 未満のときはなにもしない
A/n	001	アキュムレータの値と n 番地の内容とを加えて その結果をアキュムレータに格納する
B/n	010	アキュムレータの値から n 番地の内容を引いて その結果をアキュムレータに格納する
L/n	011	n 番地の内容をアキュムレータに格納する
T/n	100	アキュムレータの内容を n 番地に格納する
R/n	101	入力装置から入力した内容を n 番地に格納する
W/n	110	n 番地の内容を出力装置に出力する
S/n	111	アキュムレータの内容を n ビット左にシフトする 最下位の桁には 0 を埋める

[1] 付属の取扱説明書 P.15 には「ACCUMLATOR が正の時」とあるが取扱説明書の方が誤りである。

レジスタのところに書かれた数値 (2^0など) は、そのビットが表す値を示している。例えば、10 を入力したい場合には

$$10 = 8 + 2 = 2^3 + 2^1$$

であるため、右から 4 番目と 2 番目の 2 つを 1 に、それ以外を 0 にする。コンピュータでは、このように 0 と 1 を組み合わせて値を表現しており、この 0 か 1 という情報を 1 ビットと呼んでいる。スロースキャンコンピュータのように 8 つのビットを操作できるコンピュータは、8 ビットコンピュータと呼ばれる。ビットを指定する場合には、最下位のビットを 0 ビット目として数えてゆく。先ほどの 10 を指定する例では、「3 ビット目と 1 ビット目を 1 に、それ以外のビットを 0 にする」などと言う。

ちなみに、負の数は 2 の補数形式で表現する。つまり、負の数の絶対値を 2 進数で表現し、それぞれのビットを反転し、さらに 1 を加えた値が表現したい負の数の 2 進数表現である。-1 は 11111111 であり、-10 は 11110110 である。

2.3 プログラム実行の実際

実際にプログラムを組んで、スロースキャンコンピュータに入力し、実行するまでを見てみる。

2.3.1 プログラムの作成

ここでは、メモリの特定のアドレスに格納されている値を 2 倍し、それを出力するプログラムを考える。このプログラムは次のようになる。

```
L/5
A/5
T/5
W/5
J/0
```

このプログラムは、

- L/5
  5 番地に格納されている値をアキュムレータに取り出し
- A/5
  アキュムレータの値と 5 番地に格納されている値を加算して結果をアキュムレータに格納し

- T/5
  現在のアキュムレータの内容をメモリの 5 番地に書き込み
- W/5
  5 番地に格納されている値を出力レジスタに出力し
- J/0
  プログラムを終了する

というものである。また、5 番地には値の初期値を指定しておく。

次は、このプログラムを機械語に変換しなければならない。スロースキャンコンピュータでは、コード部に 3 ビット、アドレス部に 5 ビットを割り当てている。コード部は表 1に基づいて変換し、アドレス部はアドレス指定されている値を 2 進数で表現すればよい。このような変換を行うことで、先ほどのプログラムは次のように変換できる（5 番地に格納する初期値として 1 を用いている）。

```
0:  L/5   011 00101
1:  A/5   001 00101
2:  T/5   100 00101
3:  W/5   110 00101
4:  J/0   000 00000
5:        000 00001
```

2.3.2 プログラムの書き込み

プログラムの書き込みは、入力レジスタ下のスイッチ、及び$\frac{\text{READ}}{\text{WRITE}}$を用いる。

1. 最初に$\frac{\text{STOP}}{\text{RUN}}$スイッチを STOP 側に倒す
2. 書き込みを開始するアドレスとして 00000 を指定する

(a) アドレス指定を示すために$\frac{\text{ADDRESS}}{\text{DATA}}$を ADDRESS 側に倒す
(b) アドレスとして 0 を指定するために入力レジスタの下のスイッチの下位 5 ビットを 0 を表す下側に倒す
(c) 実際に値を書き込むために$\frac{\text{READ}}{\text{WRITE}}$を WRITE 側に倒す

3. 0 番地に 01100101 を書き込む

(a) データの指定を示すために$\frac{\text{ADDREEE}}{\text{DATA}}$スイッチを DATA 側に倒す
(b) 書き込むデータとして 01100101 を表すように入力レジスタの下のスイッチを設定する
(c) 実際に値を書き込むために$\frac{\text{READ}}{\text{WRITE}}$を WRITE 側に倒す

値を書き込むと、自動的にプログラムカウンタがインクリメント[2]されてプログラムカウンタが 00001 に変わる。

4. 3の操作をプログラム終了（5 番地）まで繰り返して全てのデータを書き込む

2.3.3 プログラムの実行

プログラムの実行方法には、ステップ実行と自動実行がある。ここではまずステップ実行の方法を説明し、次に自動実行の方法を述べる。

ステップ実行 ステップ実行を行う場合、次のような手順で行う。

1. $\frac{\text{STEP}}{\text{AUTO}}$スイッチを STEP 側に倒してステップ実行することを示す
2. プログラムの実行を開始することを示すために、$\frac{\text{STOP}}{\text{RUN}}$スイッチを RUN 側に倒す
3. コンピュータの状態を初期状態に戻すために CLEAR スイッチを下に倒す

ここまでの準備段階ができたら

4. 処理を 1 ステップ進めるために F/E スイッチを下に倒す
5. 更にプログラム終了まで F/E スイッチを繰り返し操作する

処理を 1 ステップ進めると取り出しフェーズもしくは実行フェーズが実行される。

取り出しフェーズ

1. プログラムカウンタがアドレスレジスタに転送される
2. アドレスレジスタに基づき、メモリの内容がメモリレジスタに転送される

[2] 値が+1 される。

3. メモリレジスタの内容がインストラクションレジスタに転送される
4. インストラクションレジスタのアドレス部がアドレスレジスタに転送される
5. プログラムカウンタがインクリメントされる
6. インストラクションレジスタのコード部がデコーダで解析される

実行フェーズ　ロード命令の場合

1. アドレスレジスタに基づき、メモリの内容がメモリレジスタに転送される
2. メモリレジスタの内容がワーキングレジスタに転送される
3. ワーキングレジスタの内容がアキュムレータに転送される

実行フェーズで行う作業はデコーダで解析されたコードごとに異なるので注意すること。ロード命令以外の命令では異なる動作をする。

自動実行　自動実行を行う場合には、$\frac{\text{STEP}}{\text{AUTO}}$スイッチを AUTO 側に倒してから$\frac{\text{STOP}}{\text{RUN}}$スイッチを RUN 側に倒す。この時コンピュータは、停止命令に出会うか、入力命令に出会うまで処理を続ける。

3　課題

2.3で述べた実験装置の操作方法を習得するため、実際にプログラムを入力して実行してみる。また、その際のコンピュータの動作を理解することを目的とする。

課題 1

- 2.3に示したプログラムを入力し、実行してみよ。この際、取り出しフェーズや実行フェーズでクロックランプの点滅に伴いどのような処理が行われるのかを調べること。
- プログラムを再度実行した場合に出力がどうなるか考え、実行して確認してみよ。

課題 2

- 次に示すプログラムは、値を入力するとその絶対値を出力するプログラムである。このプログラムをマシン後に変換してコンピュータに入力し、さまざまなデータを入力してその動作を確認せよ。

```
R/8       ↗ B/8
L/8         T/8
J/6         W/8
B/8↗        J/0
```

- 2 数を入力して、それぞれの数の絶対値の和を出力するプログラムを作成せよ。
- 入力する値の数を入力し、さらにそれぞれの値を入力すると、その絶対値の和を出力するプログラムを作成せよ。

課題 3 次に示すプログラムは何をするプログラムか、簡潔に述べよ。

```
R/9       ↗ T/9
L/9         W/9
J/4         L/8
A/8         J/1
A/9↗
```

参考文献

- スロースキャンコンピュータ 取扱説明書, デンジニア, 1993.
- スロースキャンコンピュータ サンプルプログラム集, デンジニア, 1993.
- 計算機システム入門 講義資料, (入門計算機システム, 伊藤秀男, 倉田是, 2000).

実験 7. プロセッサの基礎: 2 週目

— アセンブリ言語 —

1 実験の目的

マイクロプロセッサはコンピュータ内で基本的な演算処理を行う半導体チップである。先週の実験ではスロースキャンコンピュータを用い、マイクロプロセッサの動作について学習した。本実験では架空のコンピュータ COMET II および、アセンブラ CASL II を題材に、アセンブリ言語の記述方法について学ぶ。

2 COMET II と CASL II

2.1 コンピュータ COMET II

スロースキャンコンピュータはプログラムをデータとして記憶装置に格納し、これを順番に読み込んで実行するコンピュータであった。このようなコンピュータをノイマン型コンピュータと呼ぶ。Windows が動作する x86 プロセッサ をはじめ、現在使われているコンピュータのほとんどがノイマン型コンピュータである。

本実験で扱う COMET II もまたノイマン型コンピュータであり、情報処理技術者試験用に作られた架空のコンピュータである。COMET II は現実に使用されているコンピュータと比べれば、はるかに簡単な構造をしている。しかし、現実のノイマン型コンピュータが持っている基本的な構造をしっかり踏襲してあり、コンピュータの入門者にとっては格好の教材と言える。

2.2 スロースキャンコンピュータと COMET II

図 1, 2 にそれぞれスロースキャンコンピュータと COMET II の構造を示す。簡単のため配線は省略してある。これより、両者の構造が似ていることが分かる。Pentium シリーズをはじめ、ノイマン型コンピュータである既存のコンピュータもまたこれらと似た構造をしている。

スロースキャンコンピュータと COMET II は似た構造をとってはいるが、細かい点で異なる。COMET II のスロースキャンコンピュータとの主な相違点を以下にまとめる。各相違点の詳細については必要に応じて追って説明する。

- アキュムレータの代わりに 8 個のジェネラルレジスタ (General Register, GR)
- オーバーフローフラグレジスタの他に サインフラグ、ゼロフラグ
- スタックポインタというレジスタの追加 (図 2 右上、スタックについては本書では説明しない)
- ALU の機能強化
- 各レジスタのビット数の増加：5-8 ビット → 16 ビット
- メモリ容量の増加： 32 バイト ($= 32 \times 8$ ビット) → 65,536 ワード ($= 65,536 \times 16$ ビット)

なお、COMET II ではデータを 16 ビット単位で扱っている。この単位を本書では “ワード” と呼ぶ。

次章以降で学ぶように、スロースキャンコンピュータと COMET II とでは、プログラムの作成法も構造同様、似ているが、細かい点で異なる。

2.3 アセンブラ CASL II

コンピュータは、図 3 の右に示すような、マシン語 (機械語) と呼ばれる 0 と 1 からなるデータをプログラムとして理解、実行する。しかしマシン語は人間にとって非常に理解しにくく、記述しにくい。そこでマシン語のプログラムを作成するときは、図 3 の左に示すような、マシン語と一対一に対応ながら人間にとって理解しや

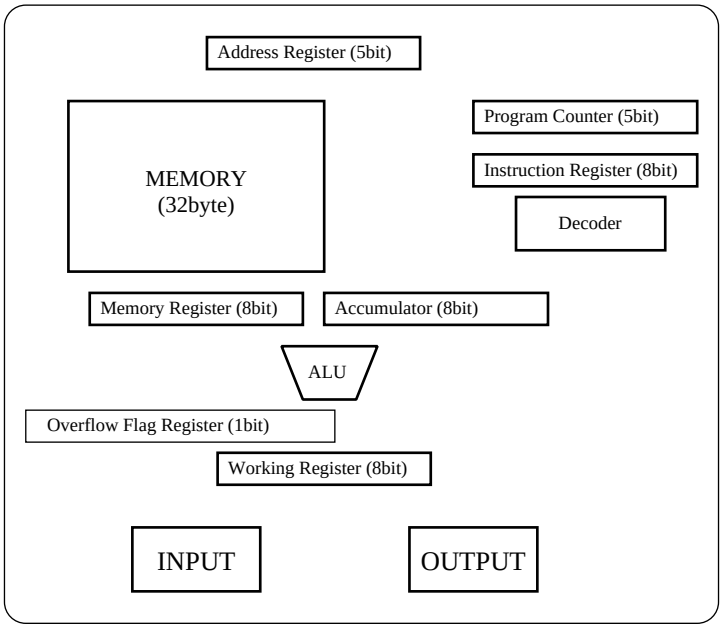

図 1: スロースキャンコンピュータ

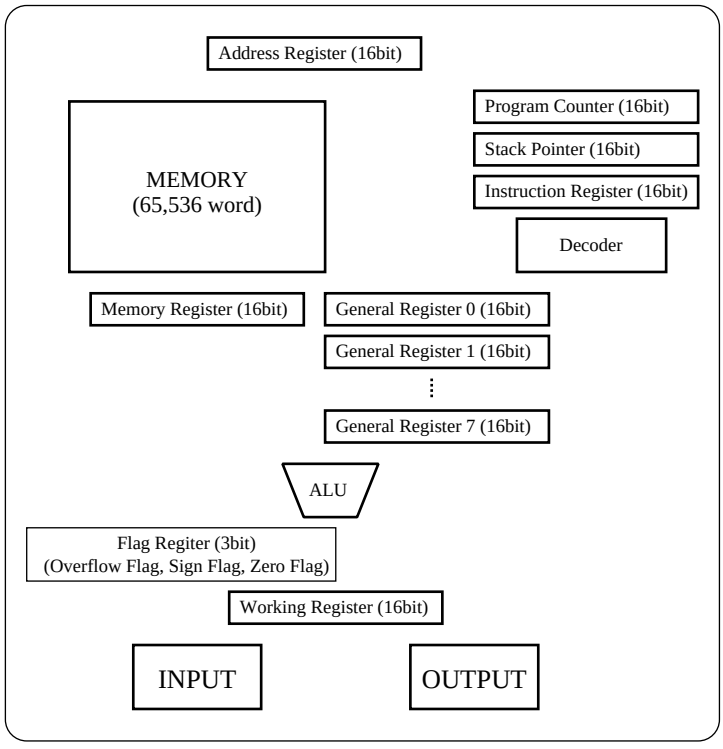

図 2: COMET II

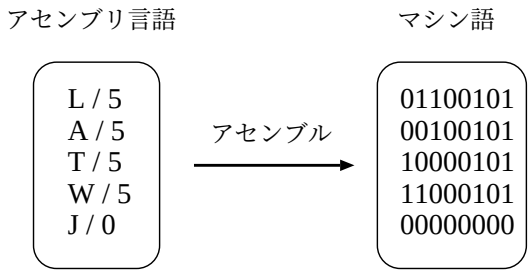

図 3: スロースキャンコンピュータのアセンブリ言語と機械語

```
EX1  START
     LD   GR0,X
     ADDA GR0,X    ; 2 倍する
     ST   GR0,X
     RET
X    DC   3
     END
```

図 4: $2X$ を求めるプログラム

すい言語を用いる。このような言語をアセンブリ言語と呼ぶ。そして、アセンブリ言語をマシン語に変換することをアセンブルすると言い、アセンブリ言語をマシン語に変換するソフトウェアをアセンブラと呼ぶ。

CASL II とは COMET II 用アセンブリ言語で記述されたプログラムを COMET II のマシン語に変換するアセンブラである。

2.4 高級言語

アセンブリ言語はマシン語と一対一対応のとれている言語であり、そのプログラミングはハードウェアを直接操作しているに等しい。そのため、アセンブリ言語によるプログラムは、機種間の移植性が非常に低く、人間にとっても理解しにくい。

現在では、アセンブリ言語はデバイスドライバや OS などハードウェアと密接に関係のあるソフトウェアを開発するときにしか用いられない。通常のプログラミングは、高級言語と呼ばれる、マシン語と一対一対応は取れていないが、移植性の高く、人間にとって理解しやすい言語を用いて行う。高級言語の例として、C 言語や Java、Perl などが挙げられる。例えば、C 言語には ANSI C という統一規格があり、規格に従って書かれたプログラムはどのようなコンピュータ上でも動作する。また、理解しやすさの違いは本実験課題において実感できるであろう。

さて、高級言語をマシン語に変換するソフトウェアとして、コンパイラとインタプリタがある。その違いは各人で調べられたい。

3 CASL II プログラミング基礎

図 4 はアセンブリ言語によるプログラムの記述例である。このプログラムは与えられた値 X について、2 倍の値 $2X$ を求めるプログラムである。本章では、このプログラムを例に CASL II によるアセンブリ言語の基礎を学ぶ。

3.1 基本文法

CASL II によるアセンブリ言語は以下の記述の繰り返しからなる。

ラベル␣命令␣オペランド␣; コメント

図 4 の例では `EX1`, `X` がラベルであり、`START`, `LD` などが命令、`GR0,X` と `3` がオペランドである。それぞれの間には空白を挿入する必要がある。

ラベルは本来数値で表されるメモリの番地を文字列で表したものである。ラベルは以下の規則を満たす範囲で自由に名付けることができる。

- 英大文字、または数字からなる
- 先頭の 1 文字は英大文字
- 8 文字以内
- 予約語 (`GR0` ~ `GR7`) 以外

ラベルは省略可能である。省略するときは行頭を空白から始める必要がある。省略しないときは行頭に空白を入れてはいけない。

アセンブリ言語によるプログラムには、“··· を (に, へ) ~ せよ” というコンピュータへの命令が繰り返されている。このうち ~ にあたる記述が “命令” であり、··· にあたる記述が “オペランド” である。例えば、スロースキャンコンピュータにおいて `L/5` は “5 番地の値を (アキュムレータへ) ロードせよ” という意味であった。

この場合、“ロードせよ” を意味している L が命令であり、“5 番地の値を” を意味している 5 がオペランドである。CASL II のアセンブリ言語においては GR0,X のようにオペランド中に “,” で区切って 2 個以上の記号を列挙することがしばしばある。このとき、記号と “,” の間に空白を入れてはいけない。

セミコロン “;” を記述することにより、それ以降、行末までにコメントを書くことができる。図 4 の例では 3 行目の “; 2 倍する” がコメントとみなされる。オペランドまたは命令とセミコロンの前には空白を入れる必要がある。

3.2 擬似命令

CASL II 用アセンブリ言語では、通常のマシン語の命令の他に、CASL II 自身に対する制御命令が 4 個定義されている。

START 命令 プログラムの開始を示す擬似命令である。すべてのプログラムは START 命令で始まる。オペランドにはプログラムの開始番地 (ラベル) を与えることができる。オペランドは省略可能である。省略した場合は START 命令の次の命令を開始位置とみなす。START 命令に対してはラベルを付ける必要がある。

END 命令 プログラムの終了を示す擬似命令である。すべてのプログラムは END 命令で終わる。オペランドはない。

DC 命令 オペランドに指定された値で初期化したメモリ領域を確保する。DC 3,4,6 のように “,” を用いて複数領域を配列のように同時に確保することもできる。

オペランドの値は 10 進数で表記する。DC #1A のように数値の直前に # 記号を付けることにより、値を 16 進数で表記することもできる。また、オペランドにラベル名を書くことができる。この場合、ラベルに対応する命令のメモリ番地で初期化する。

DS 命令 オペランドに指定された値分のメモリ領域を確保する。例えば、DS 3 は 3 ワードのメモリ領域を確保する。メモリの内容は初期化されない。

DC 命令は C 言語における初期化を含む変数宣言 (DC 3 なら int a=3;)、DS 命令は初期化しない配列の宣言 (DS 3 なら int a[3];) のようなものだと考えると理解しやすい。

3.3 ロード/ストア命令

図 4 における、LD 命令、ST 命令はそれぞれ、スロースキャンコンピュータにおける L 命令、T 命令と同様の機能を有する。

LD GR0,X においてはラベル X に対応するメモリの値をジェネラルレジスタ GR0 にロードしている。ジェネラルレジスタはスロースキャンコンピュータにおけるアキュムレータのことである。アキュムレータは 1 個しかなかったが、COMET II のジェネラルレジスタは GR0 ~ GR7 と 8 個ある。そこで、どのジェネラルレジスタにメモリの値をロードするか、オペランドで指定する必要がある。

ST GR0,X では GR0 の値をラベル X に対応するメモリにストアしている。

LD GR0,GR1 のように記述することにより、ジェネラルレジスタ間でデータを転送することもできる。

3.4 演算命令

図 4 における、ADDA 命令は加算命令である。ADDA GR0,X では GR0 にラベル X に対応するメモリの値を加算している。

COMET II は加算命令の他に以下のような演算命令をサポートしている。

- SUBA：減算命令
- AND：論理積命令
- OR：論理和命令
- XOR：排他的論理和命令

加算命令 ADDA および減算命令 SUBA の A の意味は後述する。

COMET II は否定命令 (NOT) をサポートしていない。しかし、-1 ($= 1111\cdots1$) との XOR 演算により、否定演算を行うことができる。

```
EX2  START
     LD   GR0,X
     LAD  GR1,0
     CPA  GR0,GR1 ; GR0 と 0 を比較
     JPL  L1
     SUBA GR0,X
     SUBA GR0,X
L1   ST   GR0,X
     RET
X    DC   -3
     END
```

図 5: 絶対値を求めるプログラム

3.5 RET 命令

スロースキャンコンピュータでは J/0 により、プログラムを終了した。COMET II においては RET 命令によりプログラムを終了する。

4 フラグと条件分岐

図 5 は値 X の絶対値を求めるプログラムである。本章ではこのプログラムを例に条件分岐について説明する。

スロースキャンコンピュータの J 命令はアキュムレータの値によって分岐を行うかどうか決定していた。COMET II ではジェネラルレジスタの値によっての分岐は行わない。代わりにフラグレジスタの値によって分岐を行う。

シミュレータ WCASL-II 上でプログラムを実行されたい。CASL シミュレートモードでも、COMET シミュレートモードでも良い。FR という名前の 3 ビットのレジスタが確認できる。これがフラグレジスタである。フラグレジスタは、オーバーフローフラグ、サインフラグ、ゼロフラグの 3 個のフラグからなり、それぞれ演算の結果が、桁あふれをおこしている、負の値である、あるいはゼロであるとき 1 、そうではないとき 0 となる。

プログラム実行時最初の SUBA 命令 を実行したとき、フラグレジスタが 001 となることを確認されたい。SUBA 命令が実行され、演算結果 GR0 の値が 0 となったため、ゼロフラグが 1 となる。また、同演算によって、桁あふれが起きず、負の値も得られなかったことにより、オーバーフローフラグとサインフラグは 0 となる。

フラグレジスタは演算命令および後述する比較命令において更新される。ST 命令など演算以外の命令では更新されない。LD 命令は演算命令ではないがフラグレジスタを更新する。注意されたい[1]。

3 行目 LAD GR1,0 では、GR1 に値 0 を代入している。ジェネラルレジスタに即値を代入したいときは LAD 命令を用いる。ただし、これは LAD 命令の主旨とは異なる使用法である。(説明は後述する。)

CPA は比較命令である。4 行目 CPA GR0,GR1 では GR0 と GR1 の差を求めている。しかし、減算命令 SUBA と異なり、演算結果を GR0 に代入しない。CPA 命令はフラグレジスタのみを更新する。

JPL は分岐命令である。サインフラグとゼロフラグが共に 0 であるとき、オペランドに指定された番地に分岐を行う。サインフラグとゼロフラグの一方が 1 のときには分岐しない。すなわち、直前に行われた命令 — ここでは CPA GR0,GR1 — の結果が正であるときだけ分岐を行う。

他に以下の分岐命令がある。

- JMI : サインフラグが 0 のとき分岐 (演算結果が負のとき分岐)
- JNZ : ゼロフラグが 0 のとき分岐 (演算結果が非零のとき分岐)
- JZE : ゼロフラグが 1 のとき分岐 (演算結果が零のとき分岐)
- JOV : オーバーフローフラグが 1 のとき分岐
- JUMP: フラグレジスタの値にかかわらず分岐

加算命令、減算命令、および比較命令には、算術演算と論理演算がある。算術演算の命令名

[1] これはあくまで COMET II の仕様であり、あまり他では見ない仕様である。つまり、多くのプロセッサにおいてロード命令はフラグレジスタを更新しない。本件のみならず、このような細かい仕様の差についてはそれぞれのプロセッサ毎によく確認してプログラミングする必要がある。

```
EX3  START
     LAD  GR2,0
     LAD  GR3,1
     LD   GR1,CNT
L1   SUBA GR1,GR3    ; --GR1
     ADDA GR2,TBL,GR1
     LD   GR1,GR1    ; if ( GR1 != 0 )
     JNZ  L1
     ST   GR2,SUM
     RET
CNT  DC   4
TBL  DC   4,3,8,2
SUM  DC   0
     END
```

図 6: 数列の総和を求めるプログラム

はそれぞれ ADDA, SUBA, CPA と A で終わり、論理演算の命令名はそれぞれ ADDL, SUBL, CPL と L で終わる。ジェネラルレジスタに得られる演算結果は共に等しい。しかし、フラグの変化に違いがある。算術演算は扱う数値が符号付きの数値 $(-32768 \sim 32767)$ であるとみなすのに対し、論理演算では扱う数値が符号無しの数値 $(0 \sim 65535)$ であるとみなす。例えば #4000+#4000=#8000 という加算を考える。それぞれの数値を符号付き数値であるとみなすと、この計算は $16384 + 16384 = -32768$ となり、オーバーフローを起こしている。よって、算術演算命令 ADDA 命令により、オーバーフローフラグを 1 とする。一方、符号無し数値であるとみなすと $16384 + 16384 = 32768$ となり、オーバーフローは発生していない。よって、論理演算命令 ADDL 命令により、オーバーフローフラグを 0 とする。

5 配列

図 6 は要素数 CNT なる数列 TBL の要素の総和を求めるプログラムである。本章では、このプログラムを例に配列について説明する。

5 行目 ADDA GR2,TBL,GR1 に注目されたい。この命令はメモリの $(30)_{16}$ + GR1 番地の値を GR2 に加算する。ここで、$(30)_{16}$ はラベル TBL に対応するメモリの番地 (16 進表記) である[2]。すなわち配列 TBL の GR1 番目の要素をアクセスしているとみなすことができる。第 3 オペランド、例の場合では GR1、を指標レジスタと呼ぶ。

他の演算命令、および比較命令、ロード命令、ストア命令、分岐命令でも同様に配列を扱うことができる。

第 2 オペランドには数値を書くこともできる。例えば、LD GR0,0,GR1 とすれば、0 + GR1 番地の内容を GR0 にロードする。レジスタを書くことはできない。例えば、LD GR0,GR1,GR2 とは書けない。指標レジスタに GR0 を用いることはできない。例えば、LD GR1,X,GR0 はエラーとなる。

LAD 命令についても同様の表記を行うことができる。この場合、第 2 オペランドと指標レジスタの値の和を、第 1 オペランドに指定されるレジスタに与える。LAD は Load Address の略であり、指定されるメモリのアドレスを求めるための命令である。C 言語の & (* の逆の意) を想像すると分かり易い。

参考文献

- 計算機システム序論 講義資料, (入門計算機システム, 伊藤秀男, 倉田是, 2000).

[2] 状況によっては、ラベル TBL に対応するメモリの番地が、$(30)_{16}$ とはならないかもしれない。そのときは適宜読み替えられたい。

付録 ニーモニック表

表. COMET II ニーモニック表

命令コード					書式	意味	FR
第 1 ワード				第 2 ワード			
0	0	0	0		NOP	no operation	
1	0	r	x	adr	LD r,adr[,x]	load	○
	1	r	x	adr	ST r,adr[,x]	store	
	2	r	x	adr	LAD r,adr[,x]	load address	
	4	r1	r2		LD r1,r2	load	○
2	0	r	x	adr	ADDA r,adr[,x]	add arithmetic	○
	1	r	x	adr	SUBA r,adr[,x]	substract arithmetic	○
	2	r	x	adr	ADDL r,adr[,x]	add logical	○
	3	r	x	adr	SUBL r,adr[,x]	substract logical	○
	4	r1	r2		ADDA r1,r2	add arithmetic	○
	5	r1	r2		SUBA r1,r2	substract arithmetic	○
	6	r1	r2		ADDL r1,r2	add logical	○
	7	r1	r2		SUBL r1,r2	substract logical	○
3	0	r	x	adr	AND r,adr[,x]	and	○
	1	r	x	adr	OR r,adr[,x]	or	○
	2	r	x	adr	XOR r,adr[,x]	exclusive or	○
	4	r1	r2		AND r1,r2	and	○
	5	r1	r2		OR r1,r2	or	○
	6	r1	r2		XOR r1,r2	exclusive or	○
4	0	r	x	adr	CPA r,adr[,x]	compare arithmetic	○
	1	r	x	adr	CPL r,adr[,x]	compare logical	○
	4	r1	r2		CPA r1,r2	compare arithmetic	○
	5	r1	r2		CPL r1,r2	compare logical	○
5	0	r	x	adr	SLA r,adr[,x]	shift left arithmetic	○
	1	r	x	adr	SRA r,adr[,x]	shift right arithmetic	○
	2	r	x	adr	SLL r,adr[,x]	shift left logical	○
	3	r	x	adr	SRL r,adr[,x]	shift right logical	○
6	1	0	x	adr	JMI adr[,x]	jump on minus	
	2	0	x	adr	JNZ adr[,x]	jump on non zero	
	3	0	x	adr	JZE adr[,x]	jump on zero	
	4	0	x	adr	JUMP adr[,x]	unconditional jump	
	5	0	x	adr	JPL adr[,x]	jump on plus	
	6	0	x	adr	JOV adr[,x]	jump on overflow	
7	0	0	x	adr	PUSH adr[,x]	push	
	1	r	0		POP r	pop	
8	0	0	x	adr	CALL adr[,x]	call subroutine	
	1	0	0		RET	return from subroutine	
F	0	0	x	adr	SVC adr[,x]	supervisor call	○

[] で囲った記述は省略できる。

FR が○となっている命令を実行するとフラグレジスタの値が変化する。

r, r1, r2 はジェネラルレジスタである。命令コードの値は GR0 - GR7 に対し、それぞれ 0-7 である。

adr はメモリの番地である。

x は指標レジスタである。命令コードの値は GR1 - GR7 に対し、それぞれ 1-7 である。また、指標レジスタを省略したときの命令コードは 0 となる。